INTRODUCING A SIMULATED GATEWAY FOR VALIDATION OF INDUSTRIAL COMMUNICATION IN PLC SYSTEMS

Abstract
Given the need for efficiency in the development of new devices and protocols for communication and industrial automation, simulation has become a great ally in this area. In addition, the application of tests on these new products is extremely important to avoid failures during their operation in industrial plants or substations of end customers. However, the tools and methods for applying Unit Tests to Programmable Logic Controllers (PLCs) programs are currently scarce, making it difficult to validate them before they are inserted into the plant, opening the door to inconveniences. This article presents a study on the state of the art of Unit Tests and their application in PLCs, as well as the validation of a simulated application-layer gateway that could be used for the development and testing of proprietary protocols to be used in PLCs in the future. The results show that the simulated gateway meets the expectations of the development and is able to function in the same way as a physical industrial gateway, opening doors for the development of new industrial and substation communication technologies. The authors hope that research into the state of the art of Unit Tests in their industrial applications will help other authors who are looking to learn more about this subject, aiding the continued development of technologies in this area.

1. Introduction

In 2011, the introduction of the Industry 4.0 (I4.0) concept at the Hannover Messe by Germany's leading names in industrial automation changed the way industrial automation technology developers work on their products [1]. Proposing a new industrial model based on decentralized control of production processes and collaboration between people and smart devices, I4.0 is defined by nine enabling technologies, these being Simulation, Collaborative Robotics, Systems Integration, Internet of Things (IoT), Cloud Computing, Cybersecurity, Additive Manufacturing, Big Data and Cyber-Physical Systems [1], these remaining to assist the achievement of Industry's 5.0 goals [1]. Currently, industries that are not suited to this new production model can be considered at a disadvantage, since these technologies increase productivity, flexibility and the independence of production processes [2].

Therefore, considering the need that industries and developers of new technologies face to keep up with the evolution of industrial standards and consequently of the industry [3], this article presents a simulated application layer gateway for validating Programmable Logic Controller (PLC) projects containing communication with external systems. This solution facilitates the development and validation of new protocols and industrial applications in a laboratory setting, eliminating the need to implement these projects in physical plants or devices during the engineering process. Taking advantage of this opportunity, this article will also present a survey on the state of the art of Unit Tests and their most current methodologies and tools for PLC programs, critical parts of the industrial processes, in which software faults can lead to major safety risks to life, equipment and the environment [4].

This work is organized as follows: Section 2 will introduce the simulated gateway, Section 3 will explore the state of the art on Unit Tests in PLCs, Section 4 will describe the validation of the simulated gateway, Section 5 will present the results and discussions, and Section 6 will conclude this work.

2. The Simulated Gateway

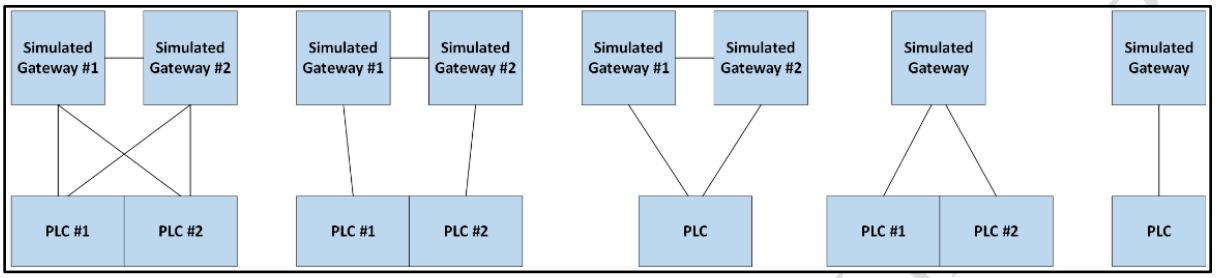
Developed in Python, the simulated application-layer gateway was idealized to support the development of an extension of a proprietary network protocol. It can receive driver messages from any driver and simulate process data. For example, the simulated gateway could be used to test the behavior of a PLC program with a new application-layer telegram or data type. The blocks can thus be developed and tested before a new physical industrial device or network is completed.

The current version can establish real Transmission Control Protocol/Internet Protocol (TCP/IP) connections to communication blocks on the PLCs and generate process values at the application layer for PLC programs. In these

programs, communication blocks are used to establish communication with external devices, using them to send process data to the driver blocks, which are used to monitor these data. Both communication and driver blocks are usually implemented as Function Blocks (FBs).

The reception of the simulated process values can be checked and confirmed with a test FB. Both the gateway and the communication blocks/PLCs can be operated in redundant or single mode. The supported operating modes in combination with the controllers are shown in Figure 1.

Figure 1 – Operation modes of the simulated gateway.

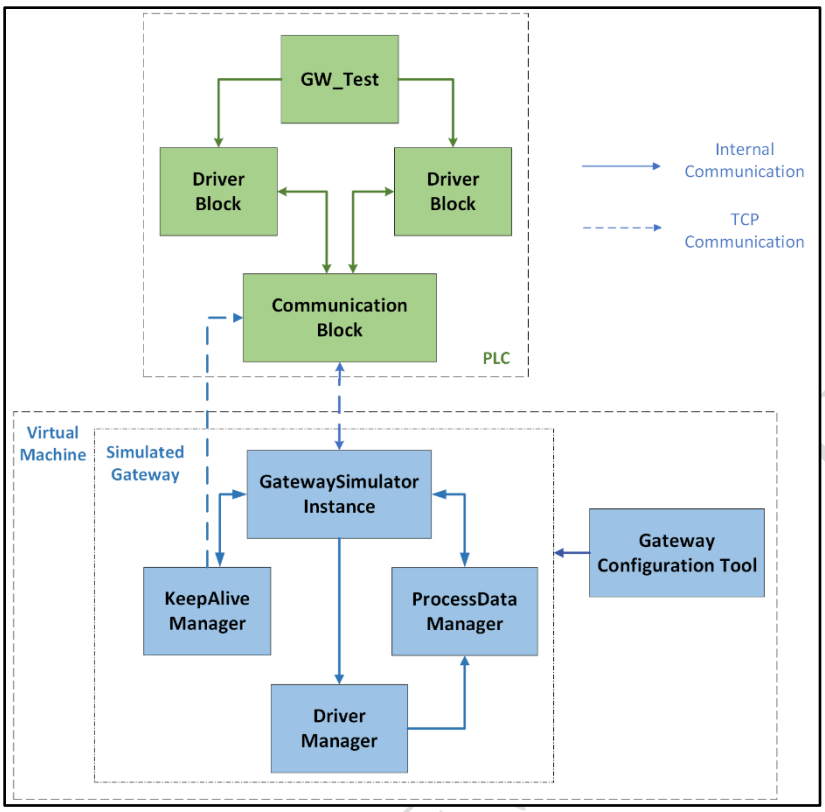


Source: The authors.

Figure 2 shows the communication paths between the different agents on the PLC and the simulated gateway sides. As the simulated gateway communicates with the PLC via TCP/IP, the corresponding ports on the environment - in this case, a Windows virtual machine (VM) - must be enabled in the firewall (both incoming and outgoing), or the firewall must be deactivated. For additional protection, the IPs of the PLCs can be authorized in the TCP rules (if necessary).

The simulated gateway system consists of the GatewaySimulator, KeepAliveManager, DriverManager and ProcessDataManager classes. The instances of the GatewaySimulator establish connections to the PLCs, receiving and sending telegrams via TCP/IP. The KeepAliveManager monitors the connections and sends KeepAlive telegrams to the communication block. The DriverManager saves the driver configurations received. The ProcessDataManager generates process data based on the driver configurations, which is sent via the gateway instances.

Figure 2 – Communication paths between the different system agents.



Source: The authors.

The GatewaySimulator instances establish a TCP connection with the PLC communication blocks. Once a connection has been established, it is added to the KeepAliveManager, DriverManager and ProcessDataManager. The KeepAliveManager monitors the transmission times of the GatewaySimulator instances. If it is determined that a KeepAlive message must be sent, because the time since the last KeepAlive telegram sent has exceeded one second, the KeepAliveManager itself sends a KeepAlive telegram via the TCP connection that was established by the GatewaySimulator instance. This is a one-way communication, which means that only KeepAliveManager sends this message according to the requests of the proprietary protocol, and does not receive any message from the communication block.

The DriverManager receives driver configuration objects - sent from the PLC - through the GatewaySimulator instance and saves them in a structured dictionary. The ProcessDataManager receives the driver configurations from the DriverManager and generates the process data telegrams based on them. These are transferred to the relevant gateway instances, which then send the telegrams to the communication block. There can be up to 4 GatewaySimulator instances per communication block, depending on the redundancy setting. Only one instance of the KeepAliveManager, DriverManager and ProcessDataManager classes can exist at once.

A new GatewaySimulator instance is created for each TCP connection, and each gateway instance is started as a new thread. Another thread is started for the send method. Each manager also runs in its own thread. A single TCP connection (simulated gateway and PLC in single mode and a communication block) therefore has 5 threads. Each additional TCP connection therefore adds two new threads. In full redundancy mode with one communication block there are 11 threads, with two communication blocks there are already 19 threads. It is therefore important to ensure that the device on which the gateway is running has enough processing power available. The simulated gateway configuration is made via its configuration tool GatewayConfigurationTool.

In this way, when using the introduced solution, the software tester can parameterize the gateway according to the test cases for the PLC program via the GatewayConfigurationTool interface. The data types to be simulated within a test can be selected there. A test contains as many runs as the number of data types selected by the user. In the first

run, process values are generated for the first data type and sent to all registered drivers until the simulation duration has expired. The next run then starts for the next data type. The TCP connections to the PLC are disconnected and reestablished again between the runs, with the registration of the communication and driver blocks. At the end of the test, there are statistics showing how often which data type was generated and sent for which driver.

The "GW_Test" test function block can test up to 16 drivers with one instance. It checks how often the drivers receive new process values. At the end of the test, the user can compare the statistics of the simulator with those of the test module and determine whether the test was successful. If it has failed, the user can see which signals have not received all the process data. The validation of this test function block will be shown in section 4.

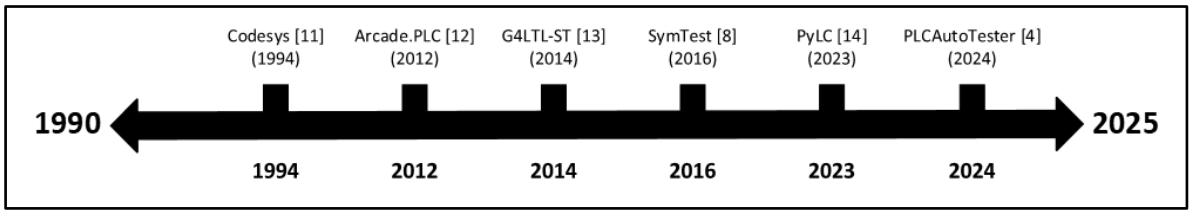
3. Unit Tests on PLCs: State of the Art, Tools and Methods

There are several ways to ensure software quality, one of which is the process of software testing. This topic includes a wide variety of testing techniques and frameworks for software in general, including those used in PLC programming. One of the most well-known of these techniques is called Unit Testing, in which every individual part of a software (units) is tested to ensure their correct operation according to their design. These units can be code pieces like functions or methods, and the testing process can be manual or automated by a framework or tool, always aiming to catch errors as soon as possible during the software development process [5]. [6], citing [7], says that 79% of Microsoft developers use unit testing in their daily work, also remembering that "unit testing is a mandatory task required in various international standards for different industrial systems, e.g. IEC 61508, ISO26262, RTCADO-178B/C, etc." [6] .

However, although unit testing is an important activity to ensure software quality, typically, this technique does not tend to be applied in PLC software, since there are few reliable automated tools available for unit test generation in this context. As a result, these unit test cases are frequently manually written by engineers, which is very time and cost demanding [6]. According to [8], "traditional testing techniques are not sufficient to detect latent errors in control system software". This limitation of traditional practices is greatly influenced by the characteristics of embedded systems, such as cyclic execution and real-time interruptions, making the testing process more difficult [8], and encouraging companies to use automated testing tools, preferably those capable of also generating the tests, not just executing them.

Therefore, to take advantage of this reality, a study was conducted to identify reliable unit test generator tools, particularly for PLC testing. A search in the Scopus database was carried out in order to find articles related to the use of software for performing tests on PLC programs in the last 5 years. This database was chosen because it is known for being comprehensive, containing more than 100 million articles from 330 disciplines [9]. The search string "TITLE-ABS-KEY ((unit AND test) AND (programmable AND logic AND controller) OR plc) AND PUBYEAR > 2019 AND PUBYEAR < 2026 AND PUBYEAR > 2019 AND PUBYEAR < 2026 AND (LIMIT-TO (SUBJAREA , 'ENGI') OR LIMIT-TO (SUBJAREA , 'COMP')) AND (LIMIT-TO (DOCTYPE , 'ar')) AND (LIMIT-TO (LANGUAGE , 'English'))" was used and returned 40 articles, but only one of them specifically addressed the topic of our research ([4]). Therefore, the snowball method [10] was used with this article, and its references were analyzed in order to better understand our technologies of interest. The criterion for this method was articles about the use of software for generating unit tests for PLCs published from the year 2020 onward and, in the absence of such articles, the consideration of articles presenting these solutions. Figure 3 shows a timeline of the release years of five software tools identified in literature during this research, which will be described in more detail below.

Figure 3 – Software found in this research and their release dates.



Source: The authors.

CODESYS: CODESYS (Controller Development System) is an Integrated Development Environment (IDE), released in 1994, designed for the programming of PLCs and other embedded systems based on the IEC 61131-3 standard. It has support for all five programming languages defined by the standard, and provides a huge variety of tools for developers, such as graphical and textual editor for code writing, library and task configuration, version control via integration with Git/SVN, hardware independence (CODESYS is hardware-agnostic) and debugging tools that can be executed without available hardware [11].

Arcade.PLC: This software is a model checker for PLCs, released in 2012, which supports both static analysis and model checking for Universal Computational Logic (∀CTL) and past-time Linear Temporal Logic (ptLTL), utilizing a Counterexample-Guided Abstraction Refinement (CEGAR) approach [12]. It is compatible with various industrial PLC languages and can analyze all types of Program Organizational Units (POUs), which are the building blocks of PLC programs [12]. Additionally, it can produce execution traces and generate corresponding test cases. A notable feature of this model is its ability to handle programs written in multiple languages, including Structured Text (ST), Instruction List (IL), and Siemens SIMATIC S7's proprietary Statement List (STL) [12]. This is achieved by compiling the input programs into an Intermediate Representation (IR), which abstracts the specific details of each language, enabling a unified analysis process [12].

G4LTL-ST: This component is a synthesis algorithm released in 2014 [13]. It is designed to generate control code for industrial PLCs based on timed behavioral specifications of input and output signals. It achieves this by applying a pseudo-Boolean abstraction to data constraints and translating timing constraints into Linear Temporal Logic (LTL). The synthesis process follows a loop based on abstraction and refinement, guided by counterstrategies. One of the tool's key strengths is its ability to produce programs in ST, compliant with the IEC 61131-3 standard. These programs can then be compiled into executable code for a wide range of industrial field-level devices. A particularly notable feature of G4LTL-ST is its support for assumption generation. When given specifications that cannot be processed, the engine can identify the causes and propose additional assumptions to make the specification feasible. It uses predefined templates and heuristic methods to do this [13]. Finally, [13] further highlights that control software synthesis, as enabled by tools like G4LTL-ST, is regarded as a foundational technology for implementing flexible and modular control systems in the context of Industry 4.0.

SymTest: SymTest is a testing framework designed for embedded systems released in 2016 [8]. It combines symbolic execution with constraint solving techniques to automatically generate test sequences. The process begins with the program's control flow graph, from which the framework identifies an optimal syntactic path. It then symbolically executes this path to produce a symbolic execution trace. From this trace, a path predicate is extracted and provided to a Satisfiability Modulo Theories (SMT) solver. If the solver determines the predicate to be satisfiable, it returns concrete values that can be used as test inputs. A major benefit of SymTest is its ability to produce minimal-length test sequences, which leads to lower testing costs and faster execution times. Additionally, the framework supports customization through user-defined heuristics [8].

PyLC: PyLC is a translation framework developed to enable automated test generation for PLC programs using the Pynguin testing tool, being released in 2023 [14]. The framework is organized into four sequential phases, each building on the previous one to support the end goal of automated testing. In the initial phase, PLC programs are converted into Python code. This transformation is guided by a set of translation rules and adheres to the IEC 61131-3 standard, which defines the structure and semantics of PLC code. Once translated, the Python code is passed to the Translation Validation module. This module verifies the correctness of the transformation using three distinct unit testing strategies to ensure that the behavior of the original PLC code is preserved. Ultimately, the validation step ensures that the converted Python code is reliable and suitable for further processing, particularly for automated test generation through Pynguin [14].

PLCAutoTester: The PLCAutoTester is the most recent tool found in our research, its release date being January 2024 [4]. The tool is an implementation of an automated unit test case generation framework, and a supporting algorithm specifically designed for Structured Text (ST) programs. Its foundational concept involves employing Dynamic Symbolic Execution (DSE) along with the idea of PLC states to create test cases for ST programs. This incorporation of PLC states concept into DSE is the key differentiating feature of PLCAutoTester. This feature allows the tool to better manage the impact of internal variables on logic branch selection across different execution cycles, which is crucial due to the cyclical execution nature of PLC programs where internal variables retain values from previous cycles. Beyond the above-mentioned features, the PLCAutoTester also presents other functionalities that are worth pointing out, such as support for multiple coverage criteria - such as statement coverage, branch coverage, and MC/DC coverage – and the capability of generating multi-cycle test cases. Finally, the framework can combine extracted program elements into an intermediate model called ST Control Flow Coverage Graph (ST-CFCG), which integrates control flow, FB structure, and coverage information [4].

4. Materials and Methods

In this validation, a gateway model for merging IEC 61850 Manufacturing Message Specification (MMS) with a proprietary communication protocol will be analyzed. The goal is to test a PLC program implemented to handle the proprietary protocol. To achieve this, simulated IEC 61850 MMS data will be generated by the gateway and sent to the PLC. The test setup is a full redundant configuration consisting of a redundant simulated gateway on a virtual machine (VM) with 16 CPU cores distributed over 2 sockets and 8 GB memory, and two Siemens AG SIMATIC S7-1515-R PLCs. The PLC program consists of one communication block containing one GW_TEST block, which in turn contains 8 driver blocks. The amount of data points each driver block contains, with their respective data type and communication style, are shown on Table 1. As this example involves a simulated gateway for integrating Intelligent Electronic Devices (IEDs) from an industrial electrical substation into a PLC system using IEC 61850 MMS, the process data to be simulated on the gateway will follow the standard's data types (Bool, Bitstring[2], Float, Int16, Int32, and Int64) and communication methods (polling and reporting). The data used for reporting communication includes, in addition to the value, the quality code in Word format and the timestamp in Date_And_Time format of an IEC 61850 data object. The data used for polling communication includes only the value and the quality code in Word format.

Table 1 – Amount of data points contained by each driver block on this test.

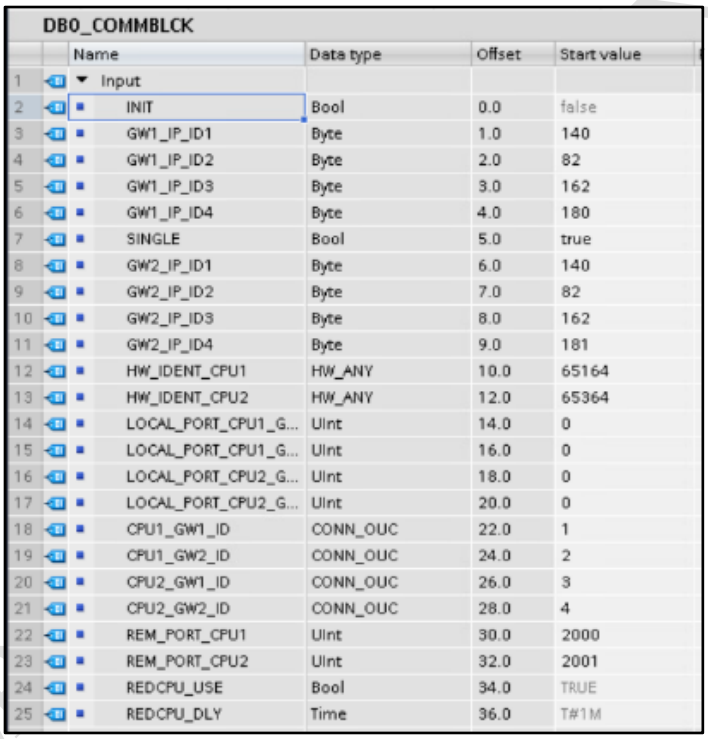
Quantity	Value Data Type	Communication
64	Bool	Reporting
10	Bitstring [2]	Reporting
10	Float	Reporting
10	Int16	Reporting
10	Int32	Reporting
10	Int64	Reporting
10	Bool	Polling
10	Bitstring [2]	Polling
10	Float	Polling
4	Int16	Polling

4	Int32	Polling
2	Int64	Polling

Source: The authors.

According to Table 1, the PLC program used in this test has 1.232 data points, since each driver block contains 154 data points and there are 8 driver blocks in the GW_TEST block on this project. For this test project, the TCP ports 2000 up to 2007 of the simulated gateway will be used. Next, the network card of the virtual machine must be configured by adding the IP addresses intended for the simulated gateway. In this case, the IP address 140.82.162.180 will be used for Gateway 1, and 140.82.162.181 for Gateway 2. The communication blocks can be configured in a cyclic interrupt organization block (OB), with a cycle of 10 ms, and the previously configured IPs of the simulated gateway must be entered in this block. In the same way, the ports specified in the firewall rules must be entered for the "REM_PORT_CPU1" and "REM_PORT_CPU2" inputs, as shown in Figure 4.

Figure 4 – Communication block configuration.



DB0_COMMBLCK

	Name	Data type	Offset	Start value
1	Input			
2	INIT	Bool	0.0	false
3	GW1_IP_ID1	Byte	1.0	140
4	GW1_IP_ID2	Byte	2.0	82
5	GW1_IP_ID3	Byte	3.0	162
6	GW1_IP_ID4	Byte	4.0	180
7	SINGLE	Bool	5.0	true
8	GW2_IP_ID1	Byte	6.0	140
9	GW2_IP_ID2	Byte	7.0	82
10	GW2_IP_ID3	Byte	8.0	162
11	GW2_IP_ID4	Byte	9.0	181
12	HW_IDENT_CPU1	HW_ANY	10.0	65164
13	HW_IDENT_CPU2	HW_ANY	12.0	65364
14	LOCAL_PORT_CPU1_G...	UInt	14.0	0
15	LOCAL_PORT_CPU1_G...	UInt	16.0	0
16	LOCAL_PORT_CPU2_G...	UInt	18.0	0
17	LOCAL_PORT_CPU2_G...	UInt	20.0	0
18	CPU1_GW1_ID	CONN_OUC	22.0	1
19	CPU1_GW2_ID	CONN_OUC	24.0	2
20	CPU2_GW1_ID	CONN_OUC	26.0	3
21	CPU2_GW2_ID	CONN_OUC	28.0	4
22	REM_PORT_CPU1	UInt	30.0	2000
23	REM_PORT_CPU2	UInt	32.0	2001
24	REDCPU_USE	Bool	34.0	TRUE
25	REDCPU_DLY	Time	36.0	T#1M

Source: The authors.

After configuring the communication blocks, the driver blocks are configured to exchange simulated process data between the simulated gateway and the PLC. The driver block should be inserted into a cyclic interrupt OB with a 60 ms cycle. Figure 5 shows the layout of the driver and GW_TEST blocks in the program.

Figure 5 – Configured driver and test blocks.

```
17 "DB1_DBLCK_6"(TestConnect := "GW_TEST_DB".TestConnect,
18               TCONNECT := "DB0_COMMBLCK".TCONNECT);
19
20 "DB1_DBLCK_7"(TestConnect := "GW_TEST_DB".TestConnect,
21               TCONNECT := "DB0_COMMBLCK".TCONNECT);
22
23 "DB1_DBLCK_8"(TestConnect := "GW_TEST_DB".TestConnect,
24               TCONNECT := "DB0_COMMBLCK".TCONNECT);
25
26
27  "GW_TEST_DB"();
28
```

Source: The authors.

To parameterize the test on the PLC side, the inputs for the associated data types can either be set to TRUE offline when the GW_TEST block is called or changed during RUN. If an input is set to FALSE, no test is performed for this data type. However, it should be noted that it is not possible to permanently change the value of an input offline set, in this way, when calling up the function block, the original value is adopted again after a CPU cycle. Figure 6 illustrates the GW_TEST block on this project. All data types were set to be tested on all instances of the GW_TEST block.

Figure 6 – Configuration of the GW_TEST block on the project.

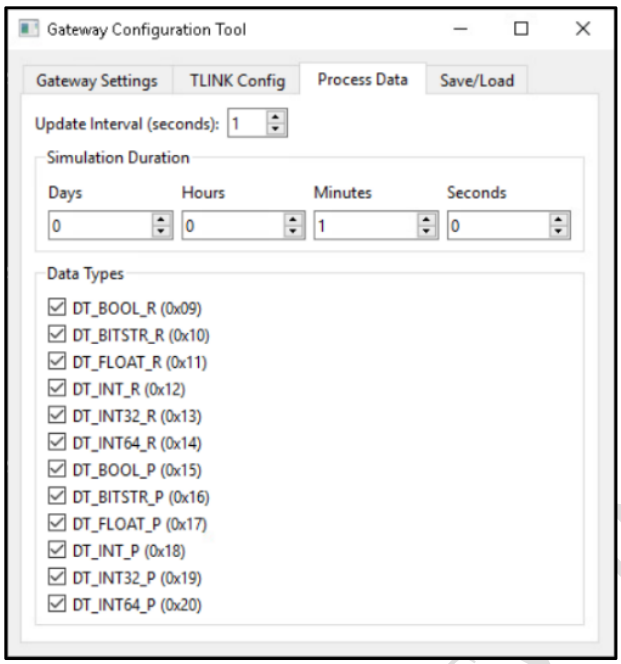
GW_TEST_DB

	Name	Data type	Offset	Start value
1	Input			
2	Reset_Counter	Bool	0.0	false
3	Bool_R	Bool	0.1	true
4	Bitstr_R	Bool	0.2	true
5	Float_R	Bool	0.3	true
6	Int_R	Bool	0.4	true
7	Int32_R	Bool	0.5	true
8	Int64_R	Bool	0.6	true
9	Bool_P	Bool	0.7	true
10	Bitstr_P	Bool	1.0	true
11	Float_P	Bool	1.1	true
12	Int_P	Bool	1.2	true
13	Int32_P	Bool	1.3	true
14	Int64_P	Bool	1.4	true

Source: The authors.

The simulated gateway is configured using the GatewayConfigurationTool. It generates a config file which is read by the gateway application. In this tool, the IP addresses for single or redundant operation modes can be configured, as well as the number of communication blocks (TCP Ports) connected to the simulation. Finally, the process data generation is also configured, as shown in Figure 7. The user can choose the desired datatypes for simulation, as well as their update interval and the duration of the simulation. So that the PLC can check the reception, the same data types as in the GW_TEST module must of course be activated here. After the configuration on the tool, the settings can be saved, and the configuration file can be loaded into the simulated gateway.

Figure 7 – Process Data generation setting in the GatewayConfigurationTool.



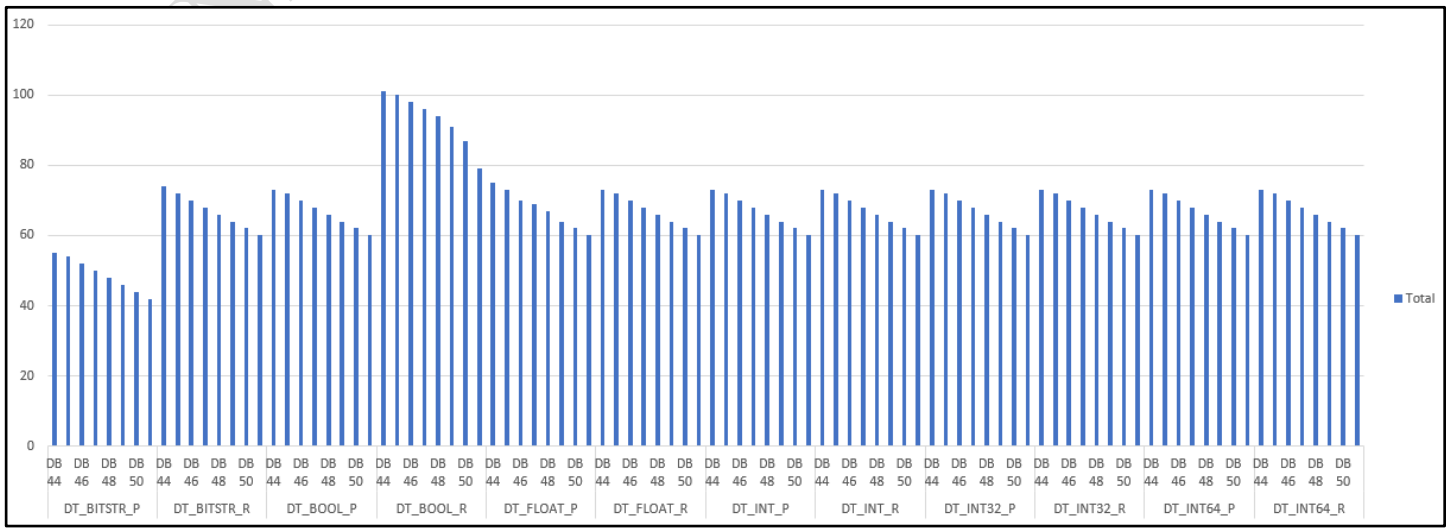
Source: The authors.

In this test each data type was simulated for 1 minute. At the end of the test, the number of generated data for each driver block is displayed in the simulation console for each simulated data type (i.e., how often process data was generated for this data type and sent to the driver block). As it takes some time to register the driver blocks in the gateway, it is possible that more data was generated for some driver blocks than for others.

5. Results and Discussion

The results of the data generated by the simulated gateway are presented in Figure 8. As shown in the image, the most generated process data was from the BOOL_R (Bool-Reporting) datatype. The figure also shows that the largest volume of data was generated for the driver block 44 (DB44), likely because it is the first DB to which the process data is sent. Additionally, it can be observed that the amount of data generated decreases over time as the tests progress, suggesting a saturation of the simulated gateway due to the computational load required during these tests.

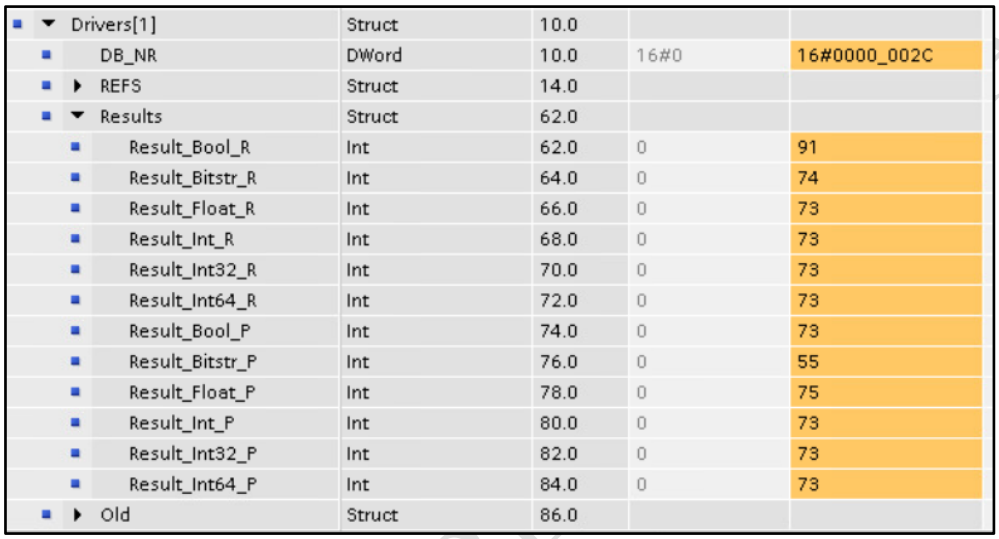
Figure 8 – Total amount of data types generated by the gateway simulator.



Source: The authors.

Examining the data received in the GW_TEST block of the PLC program reveals a different situation. As Figure 9 shows, the amount of data received on DB44 differs from the amount of data generated by the simulated gateway for this driver block. On Figure 8, the simulated gateway has generated 101 values for DB44, which has received 91 of them. This behavior was observed only for BOOL_R data types in all the project drivers, suggesting that this amount of generation load on the simulated gateway could lead to inconsistencies in the test results.

Figure 9 - Receipt of simulated data into the PLC program.



Drivers[1]	Struct	10.0		
DB_NR	DWord	10.0	16#0	16#0000_002C
REFS	Struct	14.0		
Results	Struct	62.0		
Result_Bool_R	Int	62.0	0	91
Result_Bitstr_R	Int	64.0	0	74
Result_Float_R	Int	66.0	0	73
Result_Int_R	Int	68.0	0	73
Result_Int32_R	Int	70.0	0	73
Result_Int64_R	Int	72.0	0	73
Result_Bool_P	Int	74.0	0	73
Result_Bitstr_P	Int	76.0	0	55
Result_Float_P	Int	78.0	0	75
Result_Int_P	Int	80.0	0	73
Result_Int32_P	Int	82.0	0	73
Result_Int64_P	Int	84.0	0	73
Old	Struct	86.0		

Source: The authors.

Therefore, the authors recommend using fewer driver blocks in the programs to be tested with this version of the simulated gateway, while this instability is corrected in future versions. Since the BOOL_R data type is present in massive quantities in the block, the test results were positive, as only one data type showed an inconsistency during this process.

6. Conclusion

This paper presents a simulated application-layer gateway for testing PLC open communication before it is applied in the plant. This solution is in line with the values of Industry 4.0, which encourages the use of technologies such as simulation to solve problems and innovative applications in the industrial environment. The validation results showed that the solution is capable of working with PLC systems in full redundant mode, simulating process data for driver blocks in PLCs, although it still has limitations concerning the amount of data that can be generated at once, issue that will be treated in further developments. In addition, this paper presented a short state of the art on existing unit testing technologies for PLC systems, showing their different applications. In future work, the integration of Artificial Intelligence (AI) in Engineering Tools as the ones we have presented here will be analyzed, in order to understand how AI technology can assist on test case generation for PLC programs. In addition, this work will consider virtual PLCs in the next steps. The authors hope that this article will help other researchers in the field to develop new techniques for testing programs, industrial networks, and protocols before they are applied in plants, increasing the safety and reliability of these systems.

References

[1] Xu X, Lu Y, Vogel-Heuser B, et al. Industry 4.0 and Industry 5.0—Inception, conception and perception. Journal of Manufacturing Systems. 2021;61:530–535.

[2] Javaid M, Haleem A, Singh RP, et al. Understanding the adoption of Industry 4.0 technologies in improving environmental sustainability. Sustainable Operations and Computers. 2022;3:203–217.

[3] ZVEI. Process Industrie 4.0: The Age of Modular Production—On the Doorstep to Market Launch. Frankfurt am Main, Germany: ZVEI-German Electrical and Electronic Manufacturers Association; 2019.

[4] Shi J, Chen Y, Li Q, et al. Automated Test Cases Generator for IEC 61131-3 Structured Text Based Dynamic Symbolic Execution. IEEE Trans. Comput. 2024;73:1048–1059.

[5] Wei C, Xiao L, Yu T, et al. How Do Developers Structure Unit Test Cases? An Empirical Analysis of the AAA Pattern in Open Source Projects. IIEEE Trans. Software Eng. 2025;51:1007–1038.

[6] Zhang C, Yan Y, Zhou H, et al. Smartunit. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice; New York, NY, USA; 2018; p. 296–305.

[7] Software development at microsoft observed. [place unknown]: [publisher unknown]; 2005.

[8] Suresh VP, Chakrabarti S, Jetley R. Automated Test Case Generation for Programmable Logic Controller Code. In: Proceedings of the 12th Innovations in Software Engineering Conference (formerly known as India Software Engineering Conference); New York, NY, USA; 2019; p. 1–4.

[9] Elsevier. Scopus: Content coverage guide [Internet] [cited 2025 Jul 11]. Available from: https://www.elsevier.com/products/scopus/content.

[10] Snowball sampling: A purposeful method of sampling in qualitative research. [place unknown]: [publisher unknown]; 2017.

[11] CODESYS GmbH. CODESYS – the IEC 61131-3 automation software [Internet] [cited 2025 Jul 11]. Available from: https://www.codesys.com/.

[12] Biallas S, Brauer J, Kowalewski S. Arcade.PLC: a verification platform for programmable logic controllers. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering; New York, NY, USA; 2012; p. 338–341.

[13] Cheng C-H, Huang C-H, Ruess H, et al. G4LTL-ST: Automatic Generation of PLC Programs. In; 2014; p. 541–549.

[14] Ebrahimi Salari M, Enoiu EP, Afzal W, et al. PyLC: A Framework for Transforming and Validating PLC Software using Python and Pynguin Test Generator. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing; New York, NY, USA; 2023; p. 1476–1485.